Extracting Configuration Parameter Interactions using Static Analysis

Chelsea Metcalf, Farhaan Fowze, Tuba Yavuz, and Jose Fortes

University of Florida

Problem

- Complex software systems come with a huge number of **configuration parameters**
- Understanding their impact on **run-time behavior** as well as their **interaction with other parameters** is a challenge
- Example: Apache Hadoop

Problem

- Using default settings causes performance penalties [8]
- Finding interactions between parameters can improve performance of the software
- Also aids product based modification by specifying which configuration parameters need to be tuned
- Only a few interactions have been reported in the literature [1], [2], [4], [5], [6], [11]

Contribution

- Find interacting parameters in scope of a software component using static slicing technique
  - Thin Slicing: missing control flow dependencies
  - Scalability
- Display interactions in a formalism we call **"interaction graphs"**

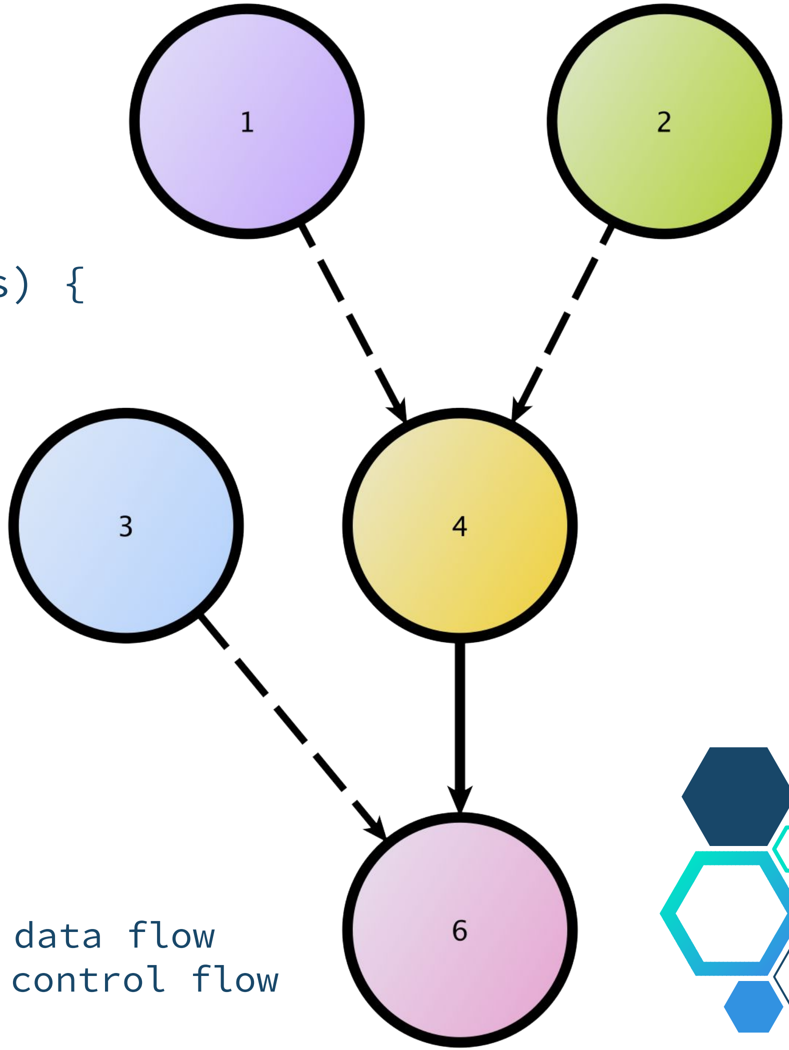
Static Analysis

- Starts by building on-the-fly call graph
  - Nodes represent methods
  - Edges represent method calls
- Our definition: Two parameters **interact** if they impact the **same program statement** through **data** and/or **control flow** dependency
- Use **WALA** program analysis framework to extract static control-flow and data-flow dependencies
- Use **Thin Slicing** to find configuration parameters that may interact

Example

```
    public void main(String[] args) {
1:    int v1 = cp.get("p1");
2:    int v2 = cp.get("p2");
3:    int v3 = cp.get("p3");
4:    if (v1 > 0 && v2 > 0)
5:      foo2(v3);
    }

    void foo2(int a1) {
6:       int f = a1;
    }
```



---> = data flow
——> = control flow

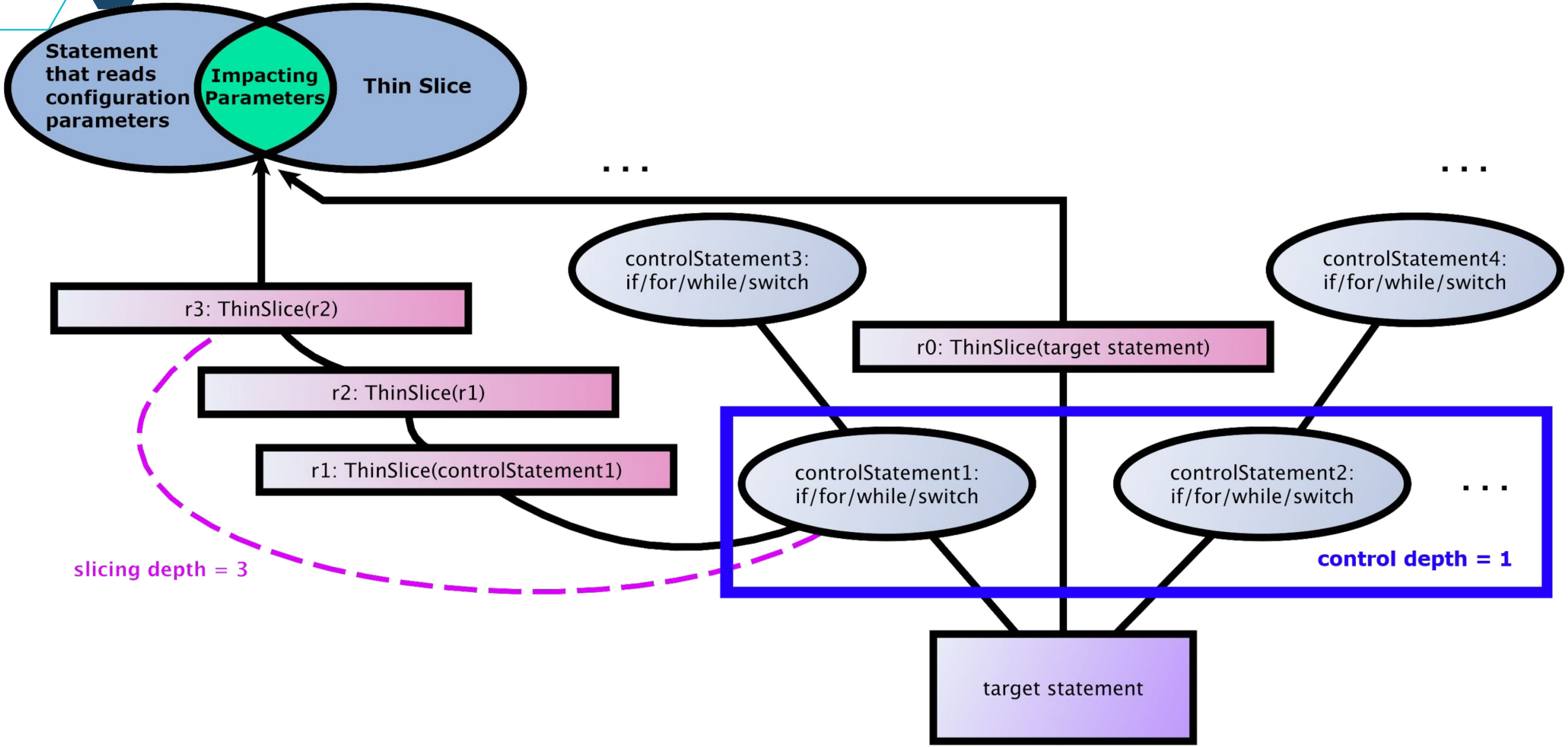
Algorithm: *FindInteractingParameters*

- <u>Input:</u> Software component
- <u>Goal:</u> Find configuration parameters that may interact
- Visit every node of call graph and every program statement in call graph node
- Find control flow statements that are controlling the target statement up to a given depth (**control depth**)

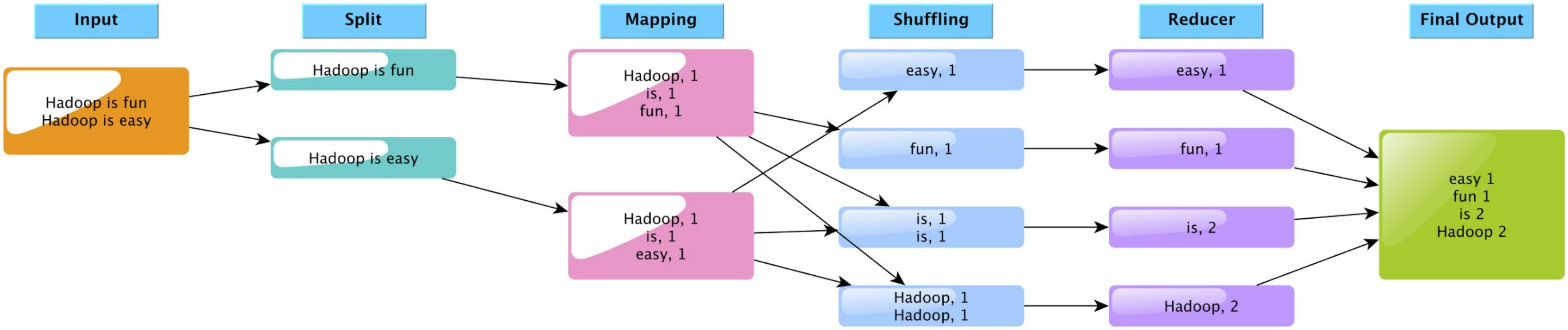
Algorithm: *FindInteractingParameters*

- Backwards Thin Slicing is performed on each statement, the **target statement**, of a given software component along with the **control statements**
- Slicing results consists of all nodes reachable from slicing seed
- Bounded data-flow dependency chain length (**slicing depth**)
- After bounded slice is computed, algorithm selects statements that represent configuration parameters and extracts names of impacting parameters

Software System Under Analysis



MapReduce Model



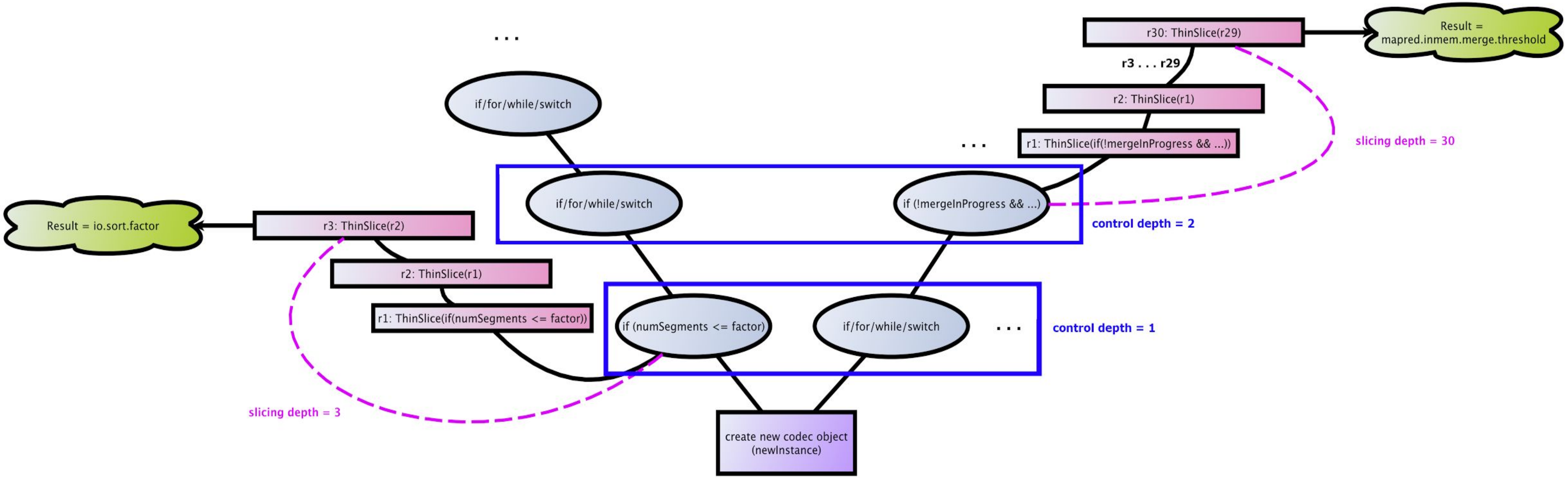
The data goes through the following phases:

- **Input Splits:** Input to a MapReduce job is divided into input splits
- **Mapping:** Data in each split is passed to a mapping function to produce output values
  - Example: WordCount, prepare a list in the form <word,frequency>
- **Shuffling:** Consolidates relevant records from Mapping phase output
- **Reducing:** Combines values from Shuffling phase and returns a single output value

Case Study

- **HADOOP-2095:** “Reducer failed due to Out of Memory”
- Hadoop was generating too many codec objects at run-time and consuming more memory than what was anticipated
- Culprit program statement in **`init()`** method of class **`SequenceFile$Reader`**:
  - `this.codec = (CompressionCodec)ReflectionUtils.newInstance(codecClass, conf);`
- **Three configuration parameters that interact:**
  - `mapred.compress.output`
  - `mapred.inmem.merge.threshold`
  - `io.sort.factor`

Case Study



<u>Input:</u> SequenceFile

<u>Seed Statement:</u> newInstance

<u>Output:</u> io.sort.factor, mapred.inmem.merge.threshold

Results

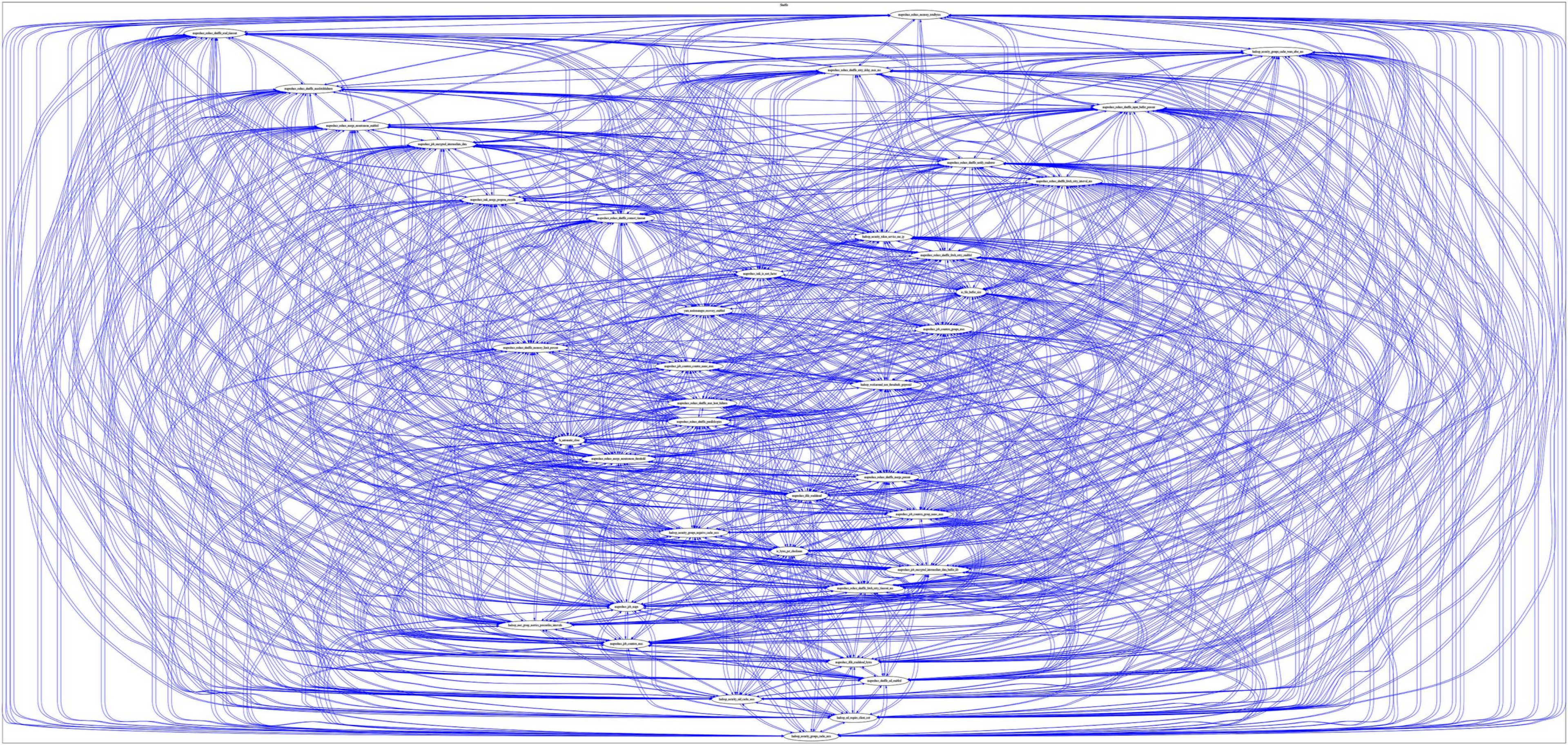
Configuration Parameter	Control Depth	Slicing Depth
`io.sort.factor`	1	3
`mapred.reduce.parallel.copies`	1	15
`mapred.reduce.copy.backoff`	1	41
`io.file.buffer.size`	1	43
`mapred.userlog.limit.kb`	1	50
`io.seqfile.compress.blocksize`	2	14
`fs.local.block.size`	2	26
`mapred.inmem.merge.threshold`	2	30
`io.sort.mb`	3	11

The set of configuration parameters that impact the codec creation in the Merge phase of the Reduce stage in **Hadoop 0.15.0**, which manifested the out of memory error as reported in **HADOOP-2095**.

Call Graph Statistics

Version	Component	# of Nodes	# of Edges	Bytecode Size (KB)	Time (hours)
2.6.0	MapTask	17749	67648	577	3.8
	Shuffle	18433	71277	537	3.3
	Merge	18264	69725	533	5.1
0.15.0	MapTask	11789	32164	399	0.4
	ReduceTask	13617	42393	410	1.6

Presenting the Data



Quantifying Interactions

- For a software component **comp**
- Parameters p_1 & p_2 that impact **comp**
- $\text{Impact}_{comp}(p_1)$ = # of program statements in **comp** that $\mathbf{p_1}$ impacts
- $\text{Impact}_{comp}(p_1, p_2)$ = # of **common** program statements in **comp** that $\mathbf{p_1}$ and $\mathbf{p_2}$ impact
- $\text{Inter}_{comp}(p_1, p_2)$ = Interaction of parameter p_1 with p_2 in the context of **comp**

$$T = Inter_{comp}(p_1, p_2) = \frac{Impact_{comp}(p_1, p_2)}{Impact_{comp}(p_1)}$$

$$p_1 \xrightarrow{T \geqslant \theta} p_2$$

$$p_1 \rightarrow p_2 \not\equiv p_2 \rightarrow p_1$$

Interaction Graph

Hadoop Version :
2.6.0

Component :
Shuffle phase
of ReduceTask

Threshold :
70%

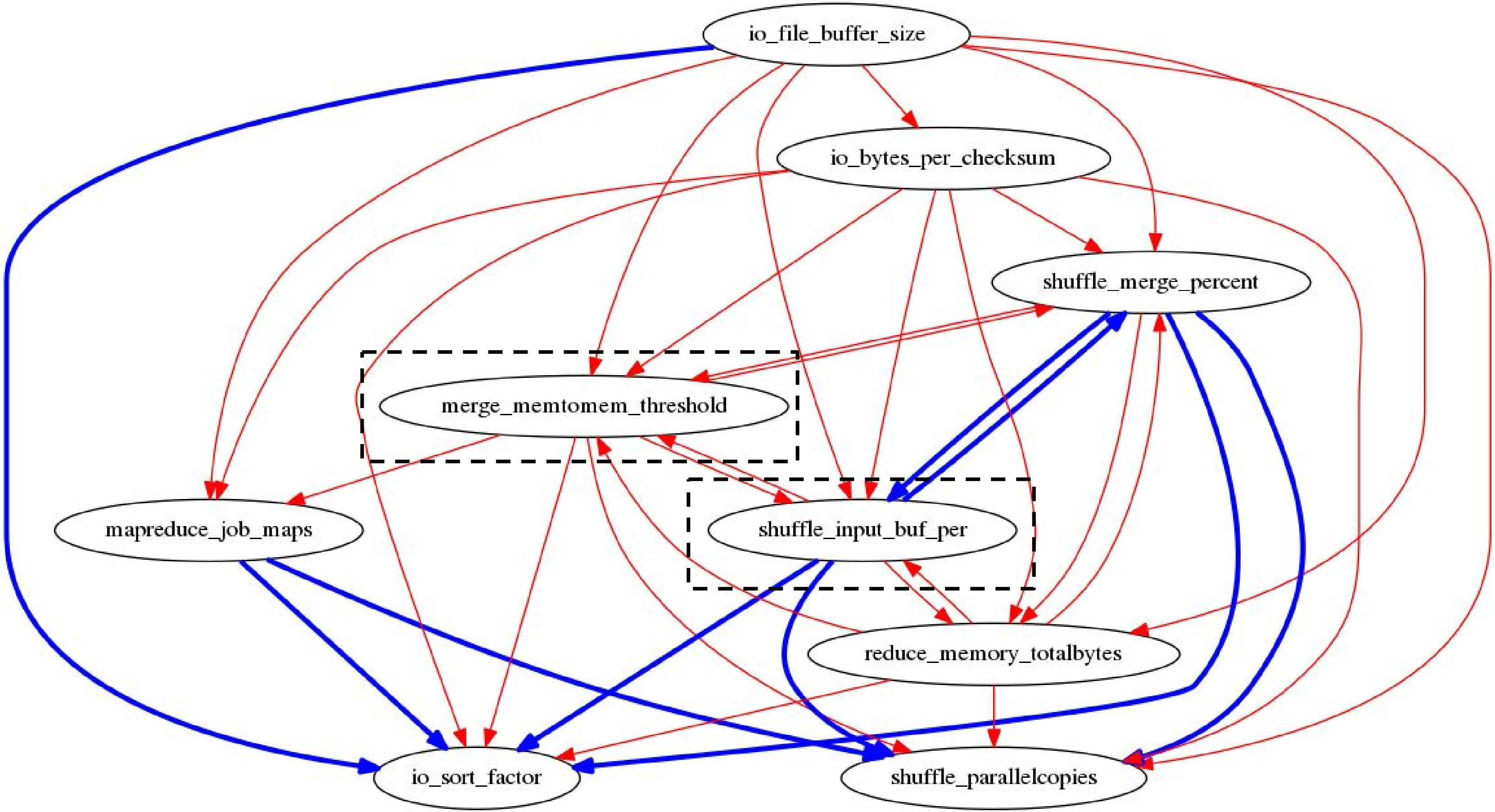
Interaction Graph Statistics

Version	Component	# of Nodes	# of Edges
2.6.0	MapTask	30	199
	Shuffle	40	146
	Merge	25	263
0.15.0	MapTask	8	15
	ReduceTask	9	37

Reported Interactions

Source	Configuration Parameter Interactions Reported in the Literature	Interaction Graph Results: Component	Interaction Graph Results: Average Interaction (InterComp)
Principal Component Analysis [22]	io.sort.factor, mapred.compress.map.output	MapTask	95%
	io.sort.mb, mapred.child.java.opts, reduce.input.buffer.percent	Merge	NA
	mapred.reduce.tasks , mapred.map.tasks, mapred.reduce.parallel.copies	Shuffle	84%
Analytical Model [11]	io.sort.mb, mapred.reduce.tasks, io.sort.spill.percent, io.sort.record.percent	MapTask	91%
	io.sort.factor, min.num.spills.for.combine	MapTask	97%
	mapred.reduce.tasks, mapred.map.tasks, mapred.job.shuffle.input.buffer.percent, mapred.child.java.opts, mapred.job.shuffle.merge.percent,io.sort.factor, mapred.inmem.merge.threshold	MapTask Shuffle	83% 95%
	mapred.job.reduce.input.buffer.percent,mapred.child.java.opts, io.sort.factor	Merge	96%
Dell's Guideline [5]	dfs.block.size, io.sort.mb, io.sort.spill.percent, io.sort.record.percent, io.file.buffer.size, io.sort.factor	DataNode MapTask	83.5% 86%
	mapreduce.tasktracker.http.threads, mapred.reduce.parallel.copies, mapred.job.shuffle.input.buffer.percent, mapred.child.java.opts, mapred.inmem.merge.threshold, mapred.job.shuffle.merge.percent,io.sort.factor, mapred.job.reduce.input.buffer.percent	Shuffle Merge	91% 93.9%
Impetus' guideline [2]	io.sort.mb, io.sort.factor	MapTask	78%
AMD's guideline [3]	io.sort.mb, io.sort.spill.percent, io.sort.record.percent	MapTask	88%

Filtered Interaction Graph



Related Work

- Precomputing possible configuration error diagnosis
  - *Ariel Rabkin and Randy Katz*
- Automated diagnosis of software configuration errors
  - *Sai Zhang and Michael Ernst*
- iTree: efficiently discovering high-coverage configurations using interaction trees
  - *Charles Song, Adam Porter, and Jeffrey Foster*

Conclusion & Future Work

- Finding configuration parameter interaction
- Our algorithm can find the most relevant parameter, although it might be missing some of the relevant ones
- Future Work:
    - Integrate static analysis with dynamic analysis to extract interactions that cross component boundaries
    - Provide more parameters for visualization

Thank You & Questions

Chelsea Metcalf: chelseametcalf@ufl.edu

Farhaan Fowze: farhaan104@ufl.edu

Tuba Yavuz: tuba@ece.ufl.edu

Jose Fortes: fortes@acis.ufl.edu

References

[1] Hadoop performance tuning by impetus. googlecode.com/files/White\%20paper-HadoopPerformanceTuning.pdf.

[2] Hadoop performance tuning guide by amd. http://www.admin-magazine.com/HPC/content/download/9408/73372/file/HadoopTuningGuide-Version5.pdf.

[3] T.J. Watson Libraries for Analysis (WALA). http://wala.sourceforge.net/wiki/index.php/Main Page.

[4] Tuning hadoop on dell poweredge servers by dell. http://www.dell.com.

[5] Herodotos Herodotu. Hadoop performance models. Technical Report CS-2011-05, Department of Computer Science, Duke University, 2000.

[6] Microsoft IT SES Enterprise Data Architect Team. Hadoop Job Optimization. Technical report, Microsoft, 2013.

[7] Ariel Rabkin and Randy Katz. Precomputing possible configuration error diagnosis. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE '11, pages 193– 202, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Kai Ren, YongChul Kwon, Magdalena Balazinska, and Bill Howe. Hadoop's adolescence: An analysis of hadoop usage in scientific workloads. Proc. VLDB Endow., 6(10):853–864, August 2013.

[9] Charles Song, Adam A. Porter, and Jeffrey S. Foster. itree: Efficiently discovering high-coverage configurations using interaction trees. IEEE Trans. Software Eng., 40(3):251–265, 2014.

[10] Manu Sridharan, Stephen J. Fink, and Rastislav Bodik. Thin slicing. SIGPLAN Not., 42(6):112–122, June 2007.

[11] Hailong Yang, Zhongzhi Luan, Wenjun Li, and Depei Qian. Mapreduce workload modeling with statistical approach. Journal of Grid Computing, 10(2):279–310, 2012.

[12] Sai Zhang and Michael D. Ernst. Automated diagnosis of software configuration errors. In Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pages 312–321, Piscataway, NJ, USA, 2013. IEEE Press.